13 | 让我们从《小汽车摇号分析》开始

2021-10-08 吴磊

《零基础入门Spark》 课程介绍 >

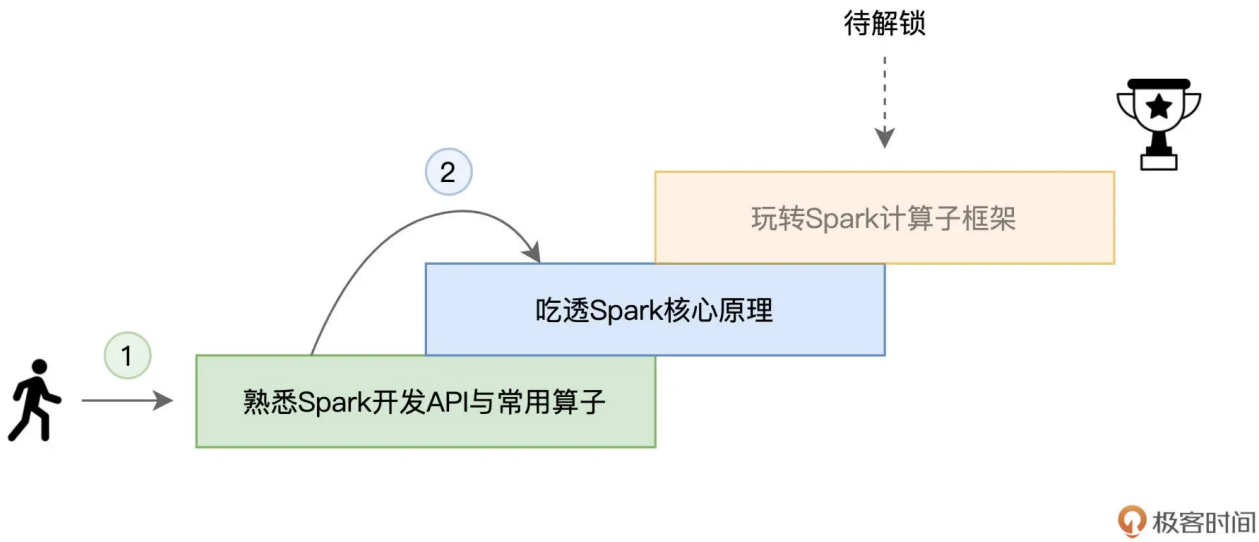
讲述：吴磊

时长 17:23 大小 15.92M

你好，我是吴磊。

在开篇词我们提出“入门 Spark 需要三步走”，到目前为止，我们携手并肩跨越了前面两步，首先恭喜你学到这里！熟练掌握了 Spark 常用算子与核心原理以后，你已经可以轻松应对大部分数据处理需求了。

不过，数据处理毕竟是比较基础的数据应用场景，就像赛车有着不同的驾驶场景，想成为 Spark 的资深赛车手，我们还要走出第三步——学习 Spark 计算子框架。只有完成这一步，我们才能掌握 Spark SQL，Structured Streaming 和 Spark MLlib 的常规开发方法，游刃有余地应对不同的数据应用场景，如数据分析、流计算和机器学习，等等。



还差第三步

那这么多子框架，从哪里入手比较好呢？在所有的子框架中，Spark SQL 是代码量最多、Spark 社区投入最大、应用范围最广、影响力最深远的那个。就子框架的学习来说，我们自然要从 Spark SQL 开始。

今天我们从一个例子入手，在实战中带你熟悉数据分析开发的思路和实现步骤。有了对 Spark SQL 的直观体验，我们后面几讲还会深入探讨 Spark SQL 的用法、特性与优势，让你逐步掌握 Spark SQL 的全貌。

业务需求

今天我们要讲的小例子，来自于北京市小汽车摇号。我们知道，为了限制机动车保有量，从 2011 年开始，北京市政府推出了小汽车摇号政策。随着摇号进程的推进，在 2016 年，为了照顾那些长时间没有摇中号码牌的“准司机”，摇号政策又推出了“倍率”制度。

所谓倍率制度，它指的是，结合参与摇号次数，为每个人赋予不同的倍率系数。有了倍率加持，大家的中签率就由原来整齐划一的基础概率，变为“**基础概率 * 倍率系数**”。参与摇号的次数越多，倍率系数越大，中签率也会相应得到提高。

不过，身边无数的“准司机”总是跟我说，其实倍率这玩意没什么用，背了 8 倍、10 倍的倍率，照样摇不上！那么今天这一讲，咱们就来借着学习 Spark SQL 的机会，用数据来为

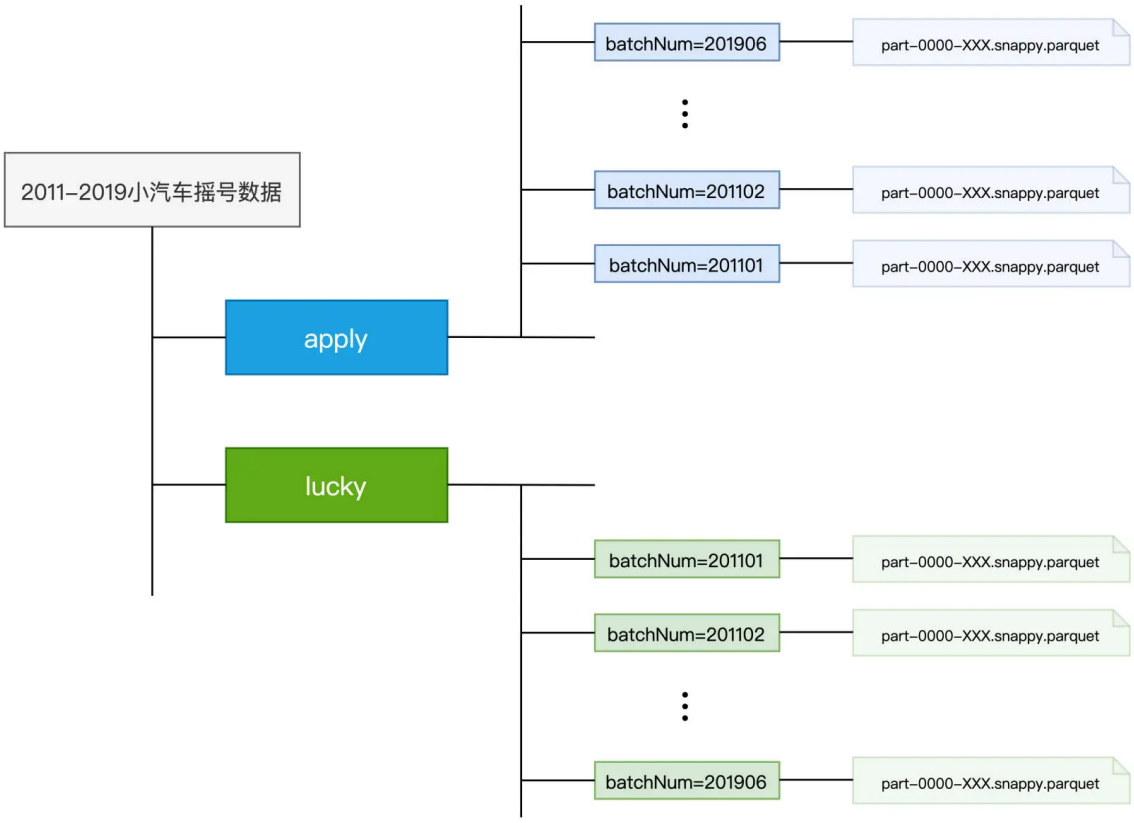
这些还没摸过车的“老司机”答疑解惑，帮他们定量地分析一下，倍率与中签率之间，到底有没有关系？

准备工作

巧妇难为无米之炊，既然是做数据分析，那咱们得先有数据才行。我这边为你准备了 2011 年到 2019 年北京市小汽车的摇号数据，你可以通过这个地址，从网盘进行下载，提取码为 ajs6。

这份数据的文件名是“2011-2019 小汽车摇号数据.tar.gz”，解压之后的目录结构如下图所示。

可以看到，根目录下有 apply 和 lucky 两个子目录，apply 目录的内容是 2011-2019 年各个批次参与摇号的申请号码，而 lucky 目录包含的是各个批次中签的申请号码。为了叙述方便，我们把参与过摇号的人叫“申请者”，把中签的人叫“中签者”。apply 和 lucky 的下一级子目录是各个摇号批次，而摇号批次目录下包含的是 Parquet 格式的数据文件。



数据的目录结构

数据下载、解压完成之后，接下来，我们再来准备运行环境。

咱们的小例子比较轻量，Scala 版本的代码实现不会超过 20 行，再者摇号数据体量很小，解压之后的 Parquet 文件总大小也不超过 4G。

选择这样的例子也是为了轻装上阵，避免你因为硬件限制而难以实验。想要把用于分析倍率的应用跑起来，你在笔记本或是 PC 上，通过启动本地 spark-shell 环境就可以。不过，如果条件允许的话，我还是鼓励你搭建分布式的物理集群。关于分布式集群的搭建细节，你可以参考第 5 讲。

好啦，准备好数据与运行环境之后，接下来，我们就可以步入正题，去开发探索倍率与中签率关系的数据分析应用啦。

数据探索

不过，先别忙着直接上手数据分析。在此之前，我们先要对数据模式（Data Schema）有最基本的认知，也就是源数据都有哪些字段，这些字段的类型和含义分别是什么，这一步就是我们常说的数据探索。

数据探索的思路是这样的：首先，我们使用 SparkSession 的 read API 读取源数据、创建 DataFrame。然后，通过调用 DataFrame 的 show 方法，我们就可以轻松获取源数据的样本数据，从而完成数据的初步探索，代码如下所示。

复制代码

```
import org.apache.spark.sql.DataFrame

val rootPath: String = _
// 申请者数据
val hdfs_path_apply: String = s"${rootPath}/apply"
// spark是spark-shell中默认的SparkSession实例
// 通过read API读取源文件
val applyNumbersDF: DataFrame = spark.read.parquet(hdfs_path_apply)
// 数据打印
applyNumbersDF.show

// 中签者数据
val hdfs_path_lucky: String = s"${rootPath}/lucky"
// 通过read API读取源文件
val luckyDogsDF: DataFrame = spark.read.parquet(hdfs_path_lucky)
// 数据打印
```

```
17 luckyDogsDF.show
```

看到这里，想必你已经眉头紧锁：“SparkSession？DataFrame？这些都是什么鬼？你好像压根儿也没有提到过这些概念呀！”别着急，对于这些关键概念，我们在后续的课程中都会陆续展开，今天这一讲，咱们先来“知其然”，“知其所以然”的部分咱们放到后面去讲。

对于 SparkSession，你可以把它理解为是 SparkContext 的进阶版，是 Spark（2.0 版本以后）新一代的开发入口。SparkContext 通过 textFile API 把源数据转换为 RDD，而 SparkSession 通过 read API 把源数据转换为 DataFrame。

而 DataFrame，你可以把它看作是一种特殊的 RDD。RDD 我们已经很熟悉了，现在就把 DataFrame 跟 RDD 做个对比，让你先对 DataFrame 有个感性认识。

先从功能分析，与 RDD 一样，DataFrame 也用来封装分布式数据集，它也有数据分区的概念，也是通过算子来实现不同 DataFrame 之间的转换，只不过 DataFrame 采用了一套与 RDD 算子不同的独立算子集。

再者，在数据内容方面，与 RDD 不同，DataFrame 是一种带 Schema 的分布式数据集，因此，你可以简单地把 DataFrame 看作是数据库中的一张二维表。

最后，DataFrame 背后的计算引擎是 Spark SQL，而 RDD 的计算引擎是 Spark Core，这一点至关重要。不过，关于计算引擎之间的差异，我们留到下一讲再去展开。

好啦，言归正传。简单了解了 SparkSession 与 DataFrame 的概念之后，我们继续来看数据探索。

把上述代码丢进 spark-shell 之后，分别在 applyNumbersDF 和 luckyDogsDF 这两个 DataFrame 之上调用 show 函数，我们就可以得到样本数据。可以看到，“这两张表”的 Schema 是一样的，它们都包含两个字段，一个是 String 类型的 carNum，另一个是类型为 Int 的 batchNum。

```
scala> applyNumbersDF.show
+------------+--------+
|       carNum|batchNum|
+------------+--------+
|1307103851262|  201906|
|3759104948944|  201906|
|1371101893066|  201906|
|3795100749475|  201906|
|1537105696290|  201906|
|3750108035493|  201906|
|1469104063180|  201906|
|3776101354768|  201906|
|1608103890895|  201906|
|3586100269069|  201906|
|1369104511811|  201906|
|3618103103088|  201906|
|1530103664336|  201906|
|3566100444078|  201906|
|1602101155778|  201906|
|3764101730069|  201906|
|1699102549506|  201906|
|3654106132739|  201906|
|1326105156204|  201906|
|3764107756542|  201906|
+------------+--------+
only showing top 20 rows
```

```
scala> luckyDogsDF.show
+------------+--------+
|       carNum|batchNum|
+------------+--------+
|7923100191899|  201401|
|0570100126544|  201401|
|6780102763770|  201401|
|4512101029666|  201401|
|8302101265951|  201401|
|1235101372069|  201401|
|6296102696988|  201401|
|0905100893507|  201401|
|2697101120908|  201401|
|5417100109768|  201401|
|2144102984814|  201401|
|1394100097074|  201401|
|8741103707804|  201401|
|2340101689355|  201401|
|1280100811950|  201401|
|7187102708637|  201401|
|6602101856993|  201401|
|9064101433730|  201401|
|9976100159260|  201401|
|5778101797083|  201401|
+------------+--------+
only showing top 20 rows
```

源数据的样本数据

其中，carNum 的含义是申请号码、或是中签号码，而 batchNum 则代表摇号批次，比如 201906 表示 2019 年的最后一批摇号，201401 表示 2014 年的第一次摇号。

好啦，进行到这里，初步的数据探索工作就告一段落了。

业务需求实现

完成初步的数据探索之后，我们就可以结合数据特点（比如两张表的 Schema 完全一致，但数据内容的范畴不同），来实现最开始的业务需求：计算中签率与倍率之间的量化关系。

首先，既然是要量化中签率与倍率之间的关系，我们只需要关注那些中签者（lucky 目录下的数据）的倍率变化就好了。而倍率的计算，要依赖 apply 目录下的摇号数据。因此，要做到仅关注中签者的倍率，我们就必须要使用数据关联这个在数据分析领域中最常见的操作。此外，由于倍率制度自 2016 年才开始推出，所以我们只需要访问 2016 年以后的数据即可。

基于以上这些分析，我们先把数据过滤与数据关联的代码写出来，如下所示。

复制代码

```
// 过滤2016年以后的中签数据，且仅抽取中签号码carNum字段
val filteredLuckyDogs: DataFrame = luckyDogsDF.filter(col("batchNum") >= "2016
// 摇号数据与中签数据做内关联，Join Key为中签号码carNum
val jointDF: DataFrame = applyNumbersDF.join(filteredLuckyDogs, Seq("carNum"),
```

在上面的代码中，我们使用 filter 算子对 luckyDogsDF 做过滤，然后使用 select 算子提取 carNum 字段。

紧接着，我们在 applyNumbersDF 之上调用 join 算子，从而完成两个 DataFrame 的数据关联。join 算子有 3 个参数，你可以对照前面代码的第 5 行来理解，这里第一个参数用于指定需要关联的 DataFrame，第二个参数代表 Join Key，也就是依据哪些字段做关联，而第三个参数指定的是关联形式，比如 inner 表示内关联，left 表示做关联，等等。

做完数据关联之后，接下来，我们再来说一说，倍率应该怎么统计。对于倍率这个数值，官方的实现略显粗暴，如果去观察 apply 目录下 2016 年以后各个批次的文件，你就会发现，所谓的倍率，实际上就是申请号码的副本数量。

比如说，我的倍率是 8，那么在各个批次的摇号文件中，我的申请号码就会出现 8 次。是不是很粗暴？因此，要统计某个申请号码的倍率，我们只需要统计它在批次文件中出现的次数就可以达到目的。

按照批次、申请号码做统计计数，是不是有种熟悉的感觉？没错，这不就是我们之前学过的 Word Count 吗？它本质上其实就是一个分组计数的过程。不过，这一次，咱们不再使用 reduceByKey 这个 RDD 算子了，而是使用 DataFrame 的那套算子来实现，我们先来看代码。

复制代码

```
val multipliers: DataFrame = jointDF.groupBy(col("batchNum"),col("carNum"))
.agg(count(lit(1)).alias("multiplier"))
```

分组计数

对照代码我给你分析下思路，我们先是用 groupBy 算子来按照摇号批次和申请号码做分组，然后通过 agg 和 count 算子把（batchNum，carNum）出现的次数，作为 carNum 在摇号批次 batchNum 中的倍率，并使用 alias 算子把倍率重命名为“multiplier”。

这么说可能有点绕，我们可以通过在 multipliers 之上调用 show 函数，来直观地观察这一步的计算结果。为了方便说明，我用表格的形式来进行示意。

batchNum	carNum	multiplier
201801	8848***2025	6
201805	8848***2025	8
201903	8848***2025	10
201702	3693***8856	2
201706	3693***8856	4
201804	3693***8856	6

极客时间

multipliers计算结果示意图

可以看到，同一个申请号码，在不同批次中的倍率是不一样的。就像我们之前说的，随着摇号的次数增加，倍率也会跟着提升。不过，这里咱们要研究的是倍率与中签率的关系，所以只需要关心中签者是在多大的倍率下中签的就行。因此，对于同一个申请号码，我们只需要保留其中最大的倍率就可以了。

需要说明的是，取最大倍率的做法，会把倍率的统计基数变小，从而引入幸存者偏差。更严谨的做法，应该把中签者过往的倍率也都统计在内，这样倍率的基数才是准确的。不过呢，结合实验，幸存者偏差并不影响“倍率与中签率是否有直接关系”这一结论。因此，咱们不妨采用取最大倍率这种更加简便的做法。毕竟，学习 Spark SQL，才是咱们的首要目标。

为此，我们需要“抹去”batchNum 这个维度，按照 carNum 对 multipliers 做分组，并提取倍率的最大值，代码如下所示。

复制代码

```
val uniqueMultipliers: DataFrame = multipliers.groupBy("carNum")
.agg(max("multiplier").alias("multiplier"))
```

分组聚合的方法跟前面差不多，我们还是先用 groupBy 做分组，不过这次仅用 carNum 一个字段做分组，然后使用 agg 和 max 算子来保留倍率最大值。经过这一步的计算之后，我们就得到了每个申请号码在中签之前的倍率系数：

carNum	multiplier
8848***2025	10
3693***8856	6

极客时间

可以看到，uniqueMultipliers 这个 DataFrame 仅包含申请号码 carNum 和倍率 multiplier 这两个字段，且 carNum 字段不存在重复值，也就是说，在这份数据集中，一个申请号码，只有一个最大倍率与之对应。

好啦，到此为止，我们拿到了每一个中签者，在中签之前的倍率系数。接下来，结合这份数据，我们就可以统计倍率本身的分布情况。

具体来说，我们想知道的是，不同倍率之下的人数分布是什么样子的。换句话说，这一次，我们要**按照倍率来对数据做分组**，然后计算不同倍率下的统计计数。不用说，这次咱们还是得仰仗 groupBy 和 agg 这两个算子，代码如下所示。

复制代码

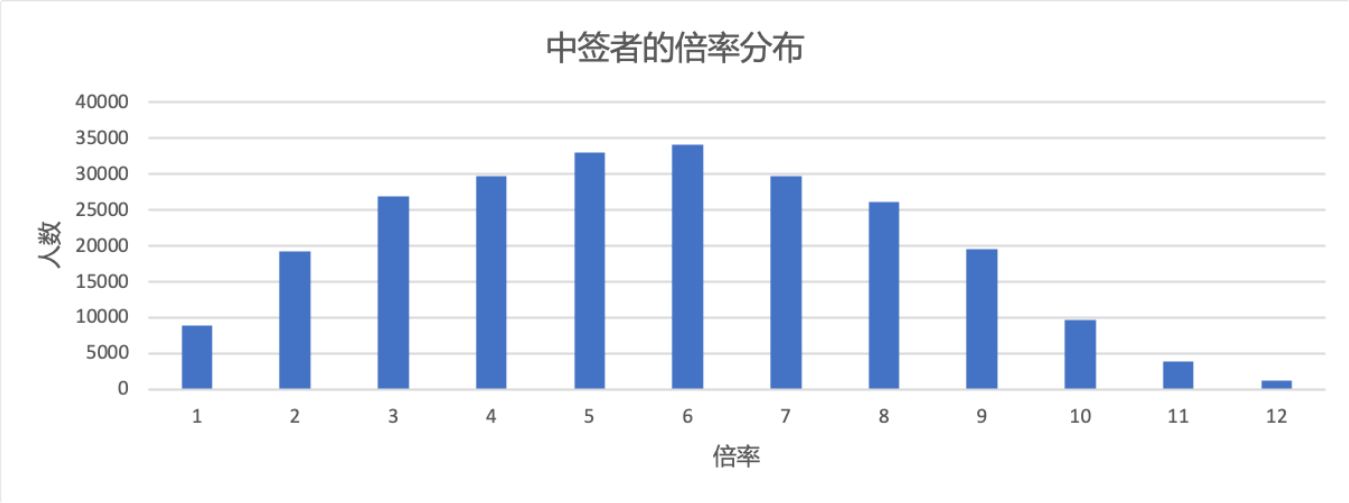
```
val result: DataFrame = uniqueMultipliers.groupBy("multiplier")
.agg(count(lit(1)).alias("cnt"))
.orderBy("multiplier")

result.collect
```

在最后一步，我们依然使用 groupBy 和 agg 算子如法炮制，得到按照倍率统计的人数分布之后，我们通过 collect 算子来收集计算结果，并同时触发上述的所有代码从头至尾交付

执行。

计算结果 result 包含两个字段，一个是倍率，一个是持有该倍率的统计人数。如果把 result 结果数据做成柱状图的话，我们可以更加直观地观察到中签率与倍率之间的关系，如下图所示。



倍率分布

不难发现，不同倍率下的中签者人数，呈现出正态分布。也即是说，对于一个申请者来说，他 / 她有幸摇中的概率，并不会随着倍率的增加而线性增长。用身边那些“老司机”的话说，中签这件事，确实跟倍率的关系不大。

重点回顾

今天这一讲，我们一起动手，开发了“倍率的统计分布”这个数据分析应用，并解答了中签率与倍率之间是否存在关联关系这一难题。

尽管在实现的过程中，我们遇到了一些新概念和新的算子，但你不必担心，更不必着急。今天这节课，你只需要对 Spark SQL 框架下的应用开发有一个感性的认识就可以了。

在 Spark SQL 的开发框架下，我们通常是通过 SparkSession 的 read API 从源数据创建 DataFrame。然后，以 DataFrame 为入口，在 DataFrame 之上调用各式各样的转换算子，如 agg、groupBy、select、filter 等等，对 DataFrame 进行转换，进而完成相应的数据分析。

为了后续试验方便，我把今天涉及的代码片段整理到了一起，你可以把它们丢进 spark-shell 去运行，观察每个环节的计算结果，体会不同算子的计算逻辑与执行结果之间的关

系。加油，祝你好运！

复制代码

```
import org.apache.spark.sql.DataFrame

val rootPath: String = _
// 申请者数据
val hdfs_path_apply: String = s"${rootPath}/apply"
// spark是spark-shell中默认的SparkSession实例
// 通过read API读取源文件
val applyNumbersDF: DataFrame = spark.read.parquet(hdfs_path_apply)

// 中签者数据
val hdfs_path_lucky: String = s"${rootPath}/lucky"
// 通过read API读取源文件
val luckyDogsDF: DataFrame = spark.read.parquet(hdfs_path_lucky)

// 过滤2016年以后的中签数据，且仅抽取中签号码carNum字段
val filteredLuckyDogs: DataFrame = luckyDogsDF.filter(col("batchNum") >= "2016

// 摇号数据与中签数据做内关联，Join Key为中签号码carNum
val jointDF: DataFrame = applyNumbersDF.join(filteredLuckyDogs, Seq("carNum"),

// 以batchNum、carNum做分组，统计倍率系数
val multipliers: DataFrame = jointDF.groupBy(col("batchNum"),col("carNum"))
.agg(count(lit(1)).alias("multiplier"))

// 以carNum做分组，保留最大的倍率系数
val uniqueMultipliers: DataFrame = multipliers.groupBy("carNum")
.agg(max("multiplier").alias("multiplier"))

// 以multiplier倍率做分组，统计人数
val result: DataFrame = uniqueMultipliers.groupBy("multiplier")
.agg(count(lit(1)).alias("cnt"))
.orderBy("multiplier")

result.collect
```

每课一练

1. 脑洞时间：你觉得，汽车摇号的倍率制度应该怎样设计，才是最合理的？

2. 请在你的 Spark 环境中把代码运行起来，并确认执行结果是否与 result 一致。

欢迎你在留言区跟我交流互动，也推荐你把这一讲的内容分享给更多的朋友、同事。我们下一讲见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 12 | 基础配置详解：有哪些配置项是你必须要关注的？

下一篇 14 | DataFrame与Spark SQL的由来

精选留言 (5)

写留言

qinsi
2021-10-12

spark无关。讨论下摇号。

评论区有匿名读者质疑文中的结论。这里尝试换个角度代入具体的数字分析下。

简单起见，假设每轮摇号有1000人中签，并且倍率和轮次一致，即第一轮大家都是1倍，...

展开

作者回复: 老弟分析得鞭辟入里~

官方的倍率设计，确实比较粗糙，统计下来，结论实际上也不是十分牢靠。咱们这里用小汽车摇号的例子，初衷还是找一个大家都熟悉的场景，来更好地学习Spark SQL的开发流程~

2

Geek_995b78
2021-10-11

用scala实现，lit(1)是什么意思呀

展开

作者回复: 实际上就是常数1，更准确地说，是表示一个常数列，这列的数值都是1。只不过Spark SQL这里的语法比较特殊，其实和Scala没什么关系哈~

1

Geek_d447af
2021-10-09

文章里的代码需要在 Hadoop 环境才能跑起来，spark 本身不支持解析 parquet 文件

作者回复: 支持的，不需要Hadoop，spark-shell本地就能跑

AIK
2021-10-08

result具体数值：
scala> result.collect
res7: Array[org.apache.spark.sql.Row] = Array([1,8967], [2,19174], [3,26952], [4,29755], [5,32988], [6,34119], [7,29707], [8,26123], [9,19476], [10,9616], [11,3930], [12,1212])

展开

作者回复: 赞👍~

Neo-dqy
2021-10-08

【.agg(count(lit(1)).alias("cnt"))】问下老师，这里count中的lit(1)是什么意思啊？
对于汽车摇号的倍率制度，如果为了优先让倍率高的人摇到号，可以把每一期的资格分多次抽取。就是说，先构建一个所有人都在里面的样本，抽部分人；再将倍率高于某个阈值的人都取出来，构建一个新的样本，再抽取部分人。（具体划分成几个样本可以按倍率的人数分布来划分）当然这样又会对新来的人不公平，所以大家还是挤地铁吧~~

展开

作者回复: lit(1)表示常数列，这列的数值都是1，是Spark SQL的语法，跟Scala无关哈~

倍率制度确实需要更好的设计~ 不过也别放弃，哈哈，万一哪天摇上了呢~